



HiSVP 量化库使用指南

文档版本 01

发布日期 2019-05-20

版权所有 © 上海海思技术有限公司2019。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <http://www.hisilicon.com/cn/>

客户服务邮箱： support@hisilicon.com



前言

概述

本文档为帮助用户了解如何使用量化库，caffe集成量化库重训的流程，以达到快速使用量化库重训的目的。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3559A	V100
Hi3559C	V100
Hi3519A	V100
Hi3516D	V300
Hi3516C	V500
Hi3559	V200
Hi3531D	V200

读者对象





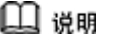
本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师
- 算法开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。



符号	说明
	用于警示紧急的危险情形，若不可避免，将会导致人员死亡或严重的人身伤害。
	用于警示潜在的危险情形，若不可避免，可能会导致人员死亡或严重的人身伤害。
	用于警示潜在的危险情形，若不可避免，可能会导致中度或轻微的人身伤害。
	用于传递设备或环境安全警示信息，若不可避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 不带安全警示符号的“注意”不涉及人身伤害。
	用于突出重要/关键信息、最佳实践和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修改描述
2019-05-20	01	第一次正式版本发布。



目 录

前言.....	i
1 概述.....	1
1.1 简介.....	1
1.2 使用场景.....	1
2 量化库.....	3
2.1 介绍.....	3
2.2 环境依赖.....	3
2.3 文件说明.....	3
2.4 接口说明.....	4
2.5 Linux 版量化库使用说明.....	5
2.5.1 检查 CUDA 版本.....	5
2.5.2 C++ 示例.....	5
2.5.3 python 示例.....	6
2.6 Windows 版量化库使用说明.....	6
3 Caffe 集成量化库.....	8
3.1 数据量化.....	8
3.1.1 下载并编译 Caffe.....	8
3.1.2 新增量化层 QuantLayer.....	8
3.1.3 链接量化库.....	9
3.2 权重量化.....	10
3.2.1 权重量化的流程.....	10
3.2.2 Finetune 训练时权重带量化的更新方法.....	11
3.2.3 网络层融合（Layer-folding）时的权重量化方法.....	12
3.3 训练.....	13
3.3.1 Finetune 训练环境的搭建.....	13
3.3.2 Finetune 训练方法及注意事项.....	13
4 FAQ.....	15
4.1 量化库报错.....	15
4.2 重训调用量化库报错.....	15



插图目录

图 1-1 NNIE 和 Caffe 数据量化反量化的位置关系.....	2
图 2-1 获取 GPU 版本结果.....	5
图 2-2 sample 运行结果.....	5
图 2-3 sample_on_python.py 运行结果.....	6
图 2-4 sample_on_python_gpu.py 运行结果.....	6
图 2-5 sample.exe 运行结果.....	6
图 3-1 量化层的位置.....	8
图 3-2 Makefile.config 修改.....	9
图 3-3 Makefile 修改(1).....	9
图 3-4 Makefile 修改(2).....	9
图 3-5 权重量化的流程.....	11
图 3-6 Finetune 训练过程.....	12
图 3-7 Batchnorm 层和 Scale 层的融合.....	13
图 4-1 量化库报错图.....	15



1 概述

1.1 简介

Caffe使用float运算，NNIE使用int8/int16运算，所以精度会有损失。设计目标是精度损失控制在1%以内，大多数网络的精度损失要小于0.5%，已经能满足应用所需。

量化算法无法保证所有情况精度损失均在期望范围内。量化精度受多种因素的影响，在某些情况下（比如待量化参数的分布离散度太高）量化精度损失可能超出设计目标。

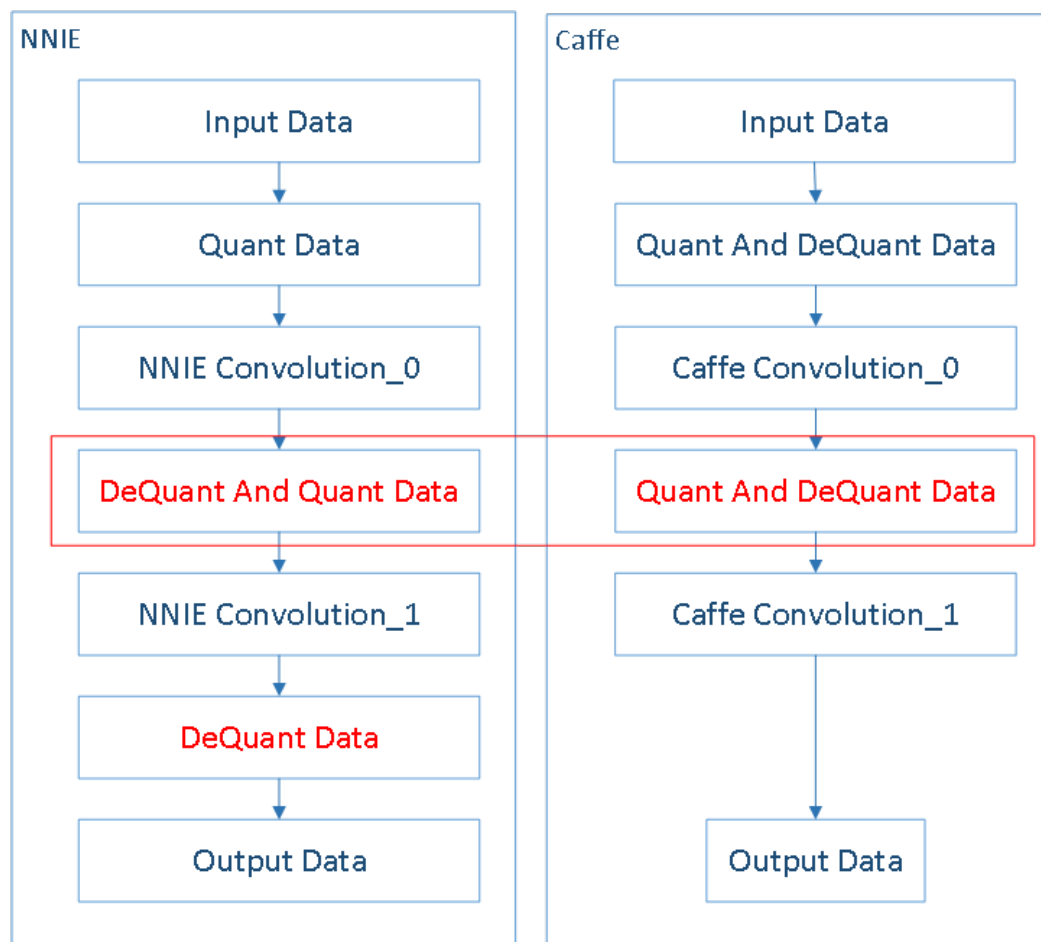
在量化精度损失不可接受的情况下，就需要带量化去重新训练以降低量化误差。

1.2 使用场景

使用NNIE 8bit/16bit运算的层主要包括Convolution、Deconvolution、DepthwiseConv、Pooling、InnerProduct，简称NNIE层。数据和权重都会被量化为8bit/16bit后，再参与运算。

所以在重训网络（Finetune）时，也需要对NNIE层的数据和权重做量化和反量化，这样就可以减少量化误差，提高精度。

图 1-1 NNIE 和 Caffe 数据量化反量化的位置关系



- NNIE的DeQuant And Quant Data 是对上一层 Convolution_0的输出做反量化，然后为下一层Convolution_1做量化。
- Caffe的Quant And DeQuant Data 是对上一层Convolution_0做量化和反量化。



2 量化库

2.1 介绍

量化库主要是提供量化反量化接口，对输入数据做量化和反量化后再输出。例如把输入的float数据量化为int8/int16，再把int8/int16反量化为float，作为输出。数据经过量化和反量化后，和原始数据的差异，即为量化误差。

2.2 环境依赖

- linux版本的编译环境：ubuntu 14.04，gcc version 4.8.5
- windows版本的编译环境：windows 10，vs2015
- CUDA版本：8.0



说明

GPU版本的量化库才依赖CUDA，CPU版本不依赖。

2.3 文件说明

HiSVP_PC_Vx.x.x.x/tools/gfpq_lib/GFPQLibs_cuda8.tgz解压后，目录结构如下：

GFPQLibs

```
|—— include
|   |—— gfpq.hpp
|—— lib
|   |—— gfpq.dll
|   |—— gfpq_gpu.dll
|   |—— gfpq_gpu.lib
|   |—— gfpq.lib
|   |—— libgfpq.a
```



```
| |—— libgfpq_gpu.a
| |—— libgfpq_gpu.so
| |—— libgfpq.so
|—— Release_note.txt
|—— sample
|—— CMakeLists.txt
|—— get_gpu_version.sh
|—— run_linux.sh
|—— run_windows.bat
|—— sample.cpp
|—— sample_on_python_gpu.py
|—— sample_on_python.py
```

- gfpq.hpp是量化库的接口头文件。
- gfpq.dll 是windows版本的动态库。
- gfpq.lib是windows版本的静态库。
- libgfpq.a是linux 版本的静态库。
- libgfpq.so 是linux 版本的动态库，软链接，指向具体的libgfpq_gpu.so.x.x.x。
- 带 “_gpu” 的库文件是GPU版本。
- Release_note.txt为量化库的版本发布说明。
- sample 目录包括量化库的使用例子。

2.4 接口说明

- HI_GFPQ_QuantAndDeQuant(): CPU版本的量化反量化接口，支持float和double类型数据，8bit和16bit量化。输入数据是CPU分配的内存。
- HI_GFPQ_QuantAndDeQuant_GPU(): GPU版本的量化反量化接口，输入数据是GPU分配的内存。
- HI_GFPQ_QuantAndDeQuant_PY(): python 或 C 的CPU接口，只支持float类型数据。
- HI_GFPQ_QuantAndDeQuant_GPU_PY(): python 或 C 的GPU接口，只支持float类型数据。
- HI_GFPQ_GetInfo(): 获取量化库的版本号和编译时间。
- GFPQ_PARAM_ST: 量化模式与量化系数。
 - GFPQ_MODE_INIT: 表示量化时不配置量化系数，接口返回传入数据生成的量化系数。
 - GFPQ_MODE_UPDATE: 表示量化时配置了量化系数，接口生成量化系数与配置的量化系数比较，使用覆盖范围更大的量化系数，并返回。
 - GFPQ_MODE_APPLY_ONLY: 表示不生成量化系数，直接使用配置的量化系数做量化反量化。



2.5 Linux 版量化库使用说明

2.5.1 检查 CUDA 版本

进入sample目录执行get_gpu_version.sh，如图2-1所示，表示服务器支持CUDA，GPU架构为sm_61。

图 2-1 获取 GPU 版本结果

```
> ./get_gpu_version.sh
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2016 NVIDIA Corporation
Built on Tue Jan 10 13:22:03 CST 2017
Cuda compilation tools, release 8.0, V8.0.61

Compile option: -gencode arch=compute_61,code=sm_61
```

量化库支持的GPU架构包括sm_30、sm_35、sm_37、sm_50、sm_52、sm_60、sm_61。

2.5.2 C++示例

解压量化库，进入sample目录，执行run_linux.sh，编译生成build-linux/sample。如果安装了CUDA驱动，还会生成sample_gpu。

执行sample，检查量化库是否能正常运行。如图2-2所示。显示了量化库的版本号和编译时间，示例数据的量化结果和性能。

具体代码参考sample.cpp。

图 2-2 sample 运行结果

```
> ./build-linux/sample
QFPQ version[1.1.5] Compiler revision[139977] Build at[Jun 2 2018 14:21:48]
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[756191]us rate[0.064571]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[756840]us rate[0.064516]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[774822]us rate[0.063019]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[742551]us rate[0.065757]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[747935]us rate[0.065284]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[790470]us rate[0.061771]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[786400]us rate[0.062091]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[794597]us rate[0.061450]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[763817]us rate[0.063926]GB/s
HI_GFPQ_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memery[ 51200]KB time[790299]us rate[0.061784]GB/s
```

注意

- Linux版本必须在linux下解压，因为量化库含软链接，如果在windows下解压，会导致软链接文件格式损坏，导致量化库不可用。
- sample 的run linux.sh依赖cmake 2.6以上。

2.5.3 python 示例

执行sample on python.py 如图2-3。

图 2-3 sample_on_python.py 运行结果

```
> python sample_on_python.py
Origin data: 0.0000000000 1.0000000000 2.0000000000 3.0000000000 4.0000000000 5.0000000000 6.0000000000 7.0000000000 8.0000000000 9.0000000000
QuantAndDeQuant: 0.0000000000 1.0000000000 2.0000000000 2.9536523819 4.0000000000 4.9674310684 5.9073047638 7.0250086784 8.0000000000 8.7240619659
Data expected: 0.0000000000 1.0000000000 2.0000000000 2.9536523819 4.0000000000 4.9674310684 5.9073047638 7.0250086784 8.0000000000 8.7240619659
HI_GFPQ_QuantAndDeQuant_PY time: 0:00:00.000053
HI_GFPQ_QuantAndDeQuant success
GFPQ_PARAM: 0x60 0x62 -0x7e 0x69 -0x4d -0x1 -0x1 -0x1 -0x3b -0x39 0x27 -0x34 0x3a 0x38 -0x28 0x33
```

执行sample on python gpu.py 如图2-4。

图 2-4 sample_on_python_gpu.py 运行结果

```
> python sample_on_python_gpu.py
Origin data: 0.0000000000 1.0000000000 2.0000000000 3.0000000000 4.0000000000 5.0000000000 6.0000000000 7.0000000000 8.0000000000 9.0000000000
QuantAndDeQuant: 0.0000000000 1.0000000000 2.0000000000 2.9536523819 4.0000000000 4.9674310684 5.9073047638 7.0250086784 8.0000000000 8.7240619659
Data expected: 0.0000000000 1.0000000000 2.0000000000 2.9536523819 4.0000000000 4.9674310684 5.9073047638 7.0250086784 8.0000000000 8.7240619659
HI_GFQP_QuantAndDeQuant_PY time:0:00.001461
HI_GFQP_QuantAndDeQuant success
GFQP PARAM: 0x60 0x62 -0x7e 0x69 -0x4d -0x1 -0x1 -0x1 -0x3b -0x39 0x27 -0x34 0x3a 0x38 -0x28 0x33
```

2.6 Windows 版量化库使用说明

解压量化库压缩包，打开DOS命令提示符，进入sample目录，执行run_windows.bat，生成build-windows\Release\sample.exe。如果支持CUDA驱动，还会生成sample_gpu.exe。sample.exe执行结果如图2-5。

图 2-5 sample.exe 运行结果

```
QFPQ version[1.1.5] Compiler revision[139977] Build at[Mar 25 2019 14:38:51]
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori      : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant   : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori      : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant   : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori      : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant   : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
Quant and dequant bit_width[8]
[quant_and_dequant][172] Data_ori      : 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
[quant_and_dequant][202] Data_dequant   : 0.000000 1.000000 2.000000 2.953652 4.000000 4.967431 5.907305 7.025009 8.000000 8.724062
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[172000]us rate[0.283884]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[157000]us rate[0.311007]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[172000]us rate[0.283884]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[172000]us rate[0.283884]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[172000]us rate[0.283884]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[141000]us rate[0.346299]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[172000]us rate[0.283884]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[188000]us rate[0.259724]GB/s
HI_GFPO_QuantAndDeQuant bit_width[ 8] cnt[ 13107200] memory[ 51200]KB time[156000]us rate[0.313001]GB/s
```



注意

sample 的run_windows.bat依赖cmake 2.6以上，依赖VS2015以上。

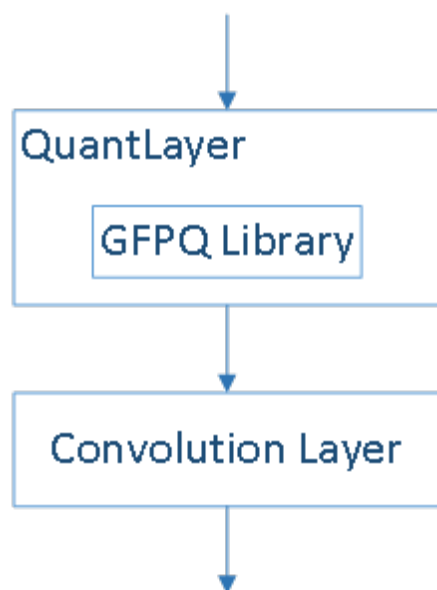
3 Caffe 集成量化库

Caffe在训练量化模型时，网络中的数据和权重都需要做量化和反量化。

3.1 数据量化

当需要新增量化层时，在量化层中调用量化库的接口做数据量化和反量化。

图 3-1 量化层的位置



3.1.1 下载并编译 Caffe

下载官方Caffe源码<https://github.com/BVLC/caffe>，先不集成量化库，确保原生Caffe可以编译通过。

3.1.2 新增量化层 QuantLayer

为在Caffe中新增量化层，可参考压缩包caffe_patch.tgz，需新增和修改的文件如下所示，新增文件包括：

- include/caffe/layers/quant_layer.hpp
- src/caffe/layers/quant_layer.cpp
- src/caffe/layers/quant_layer.cu

需修改的文件包括原始caffe环境中的src/caffe/proto/caffe.proto文件，对其进行增量修改，增加与QuantLayer相关的参数设置。

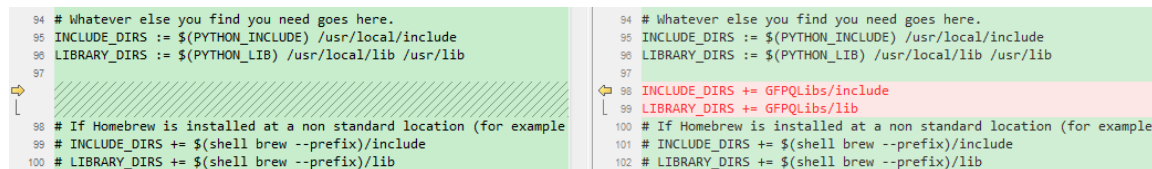
在caffe的网络结构文件prototxt中新增量化层时，需添加的内容如下所示：

```
layer {
  name: 'quant_layer_sample'
  type: 'Quant'
  bottom: 'data'
  top: ' quant_layer_sample '
  quant_param {
    quant_bits: 8
    quant_en: 1
  }
}
```

3.1.3 链接量化库

- 解压到Caffe目录，并命名为GFPQLibs。
- 修改Makefile.config，在LIBRARY_DIRS 赋初值之后，增加GFPQLibs的头文件和库路径，如[图3-2](#)所示。

图 3-2 Makefile.config 修改

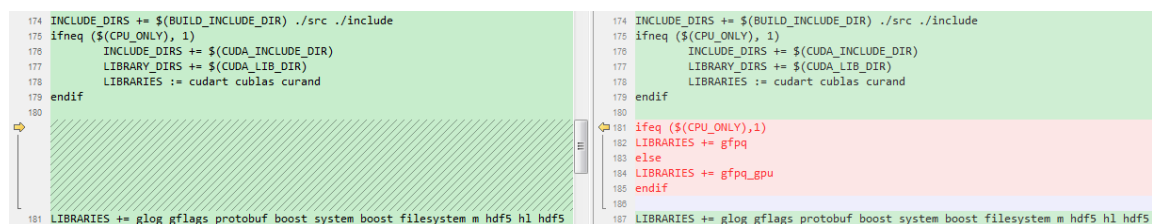


```
# Whatever else you find you need goes here.
95 INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include
96 LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib
97
98 # If Homebrew is installed at a non standard location (for example
99 # INCLUDE_DIRS += $(shell brew --prefix)/include
100 # LIBRARY_DIRS += $(shell brew --prefix)/lib
```

```
# Whatever else you find you need goes here.
95 INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include
96 LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib
97
98 INCLUDE_DIRS += GFPQLibs/include
99 LIBRARY_DIRS += GFPQLibs/lib
100 # If Homebrew is installed at a non standard location (for example
101 # INCLUDE_DIRS += $(shell brew --prefix)/include
102 # LIBRARY_DIRS += $(shell brew --prefix)/lib
```

- 修改Makefile，增加链接量化库，如[图3-3](#)所示。

图 3-3 Makefile 修改(1)



```
174 INCLUDE_DIRS += $(BUILD_INCLUDE_DIR) ./src ./include
175 ifneq ($(CPU_ONLY), 1)
176 INCLUDE_DIRS += $(CUDA_INCLUDE_DIR)
177 LIBRARY_DIRS += $(CUDA_LIB_DIR)
178 LIBRARIES := cudart cublas curand
179 endif
180
181 LIBRARIES += glog gflags protobuf boost_system boost_filesystem m hdf5_hl hdf5
```

```
174 INCLUDE_DIRS += $(BUILD_INCLUDE_DIR) ./src ./include
175 ifneq ($(CPU_ONLY), 1)
176 INCLUDE_DIRS += $(CUDA_INCLUDE_DIR)
177 LIBRARY_DIRS += $(CUDA_LIB_DIR)
178 LIBRARIES := cudart cublas curand
179 endif
180
181 ifeq ($(CPU_ONLY),1)
182 LIBRARIES += gfpq
183 else
184 LIBRARIES += gfpq_gpu
185 endif
186
187 LIBRARIES += glog gflags protobuf boost_system boost_filesystem m hdf5_hl hdf5
```

- 修改Makefile，拷贝量化库到Caffe的build目录，如[图3-4](#)所示。

图 3-4 Makefile 修改(2)



```
455 all: lib tools examples
456
457 lib: $(STATIC_NAME) $(DYNAMIC_NAME)
458
459 everything: $(EVERYTHING_TARGETS)
```

```
455 all: lib tools examples
456
457 lib: $(STATIC_NAME) $(DYNAMIC_NAME)
458 cp -af GFPQLibs/lib/* $(LIB_BUILD_DIR)/
459 everything: $(EVERYTHING_TARGETS)
```



- 执行make clean; make pycaffe -j重新编译Caffe。

3.2 权重量化

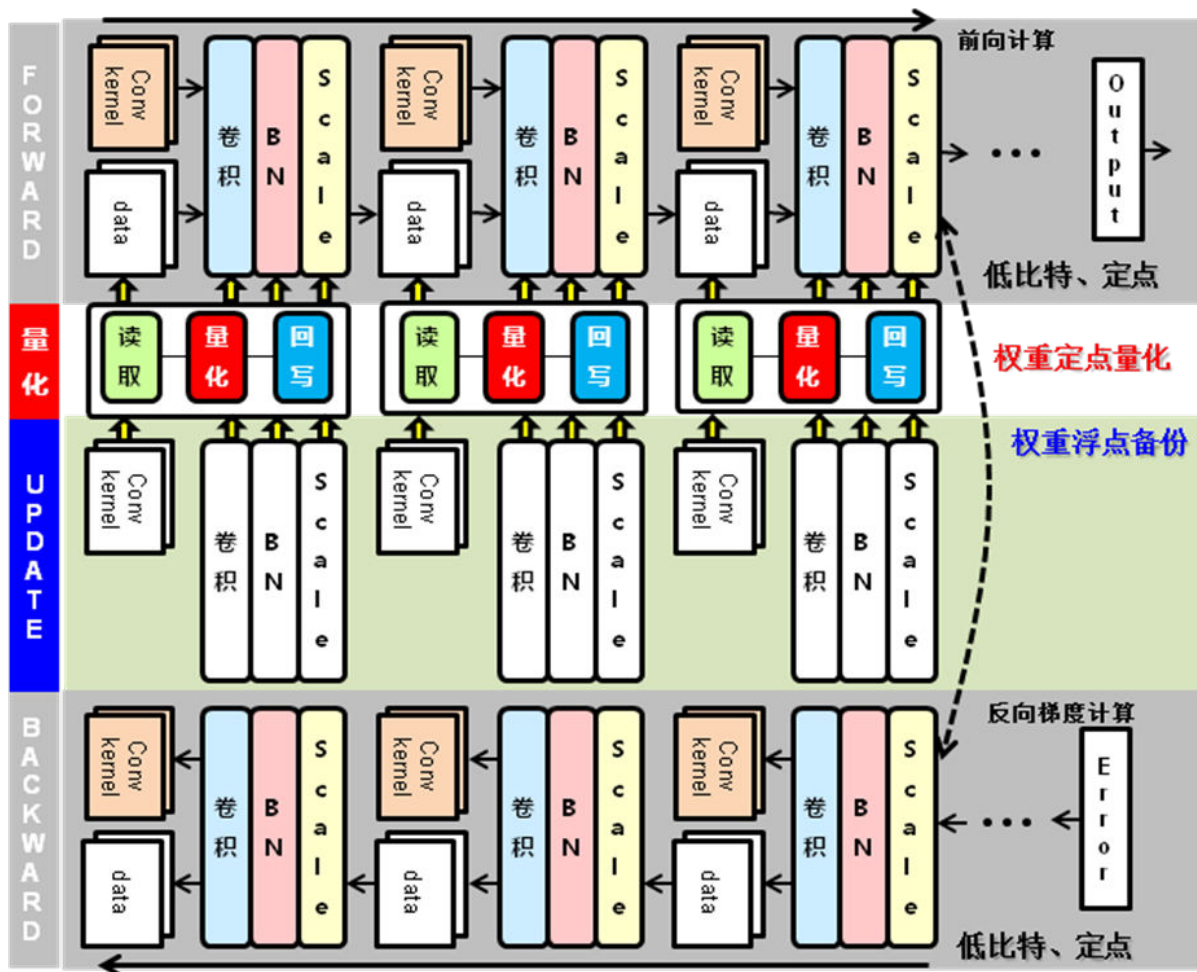
权重量化需要根据具体的训练环境在训练脚本里手动进行相应的修改。

3.2.1 权重量化的流程

卷积神经网络的训练过程大致可以概括为下面的步骤：

- 步骤1** 对网络的权重进行随机初始化。
 - 步骤2** 对网络做前向计算，即将输入数据经过卷积、Batchnorm，Scale、激活函数、池化等层后，得到网络的输出值。
 - 步骤3** 计算网络的输出值与期望的目标值（Ground Truth）之间的误差。
 - 步骤4** 当误差大于期望值时，将误差反传回网络中，依次求得池化、激活函数、Scale，Batchnorm、卷积等网络层的误差。
 - 步骤5** 根据各层误差对各网络层的梯度进行计算，再由该梯度去更新特定网络层的参数（例如卷积层的权重）。
 - 步骤6** 重复**步骤2**到**步骤5**，反复循环迭代直到网络收敛到期望的目标为止。
- 结束

图 3-5 权重量化的流程



在深度学习的众多优化算法中，典型的权重优化方法是基于随机梯度下降(Stochastic gradient descent)的动量更新(Momentum update)算法，具体公式如下：

$$w_{iter+1} = w_{iter} + \Delta history_{iter+1}$$

其中 $\Delta history$ 为网络训练过程中到当前迭代为止的所有历史梯度的累加：

$$\Delta_{iter+1} = momentum \cdot \Delta history_{iter} + learnRate \cdot \frac{\partial loss}{\partial w}$$

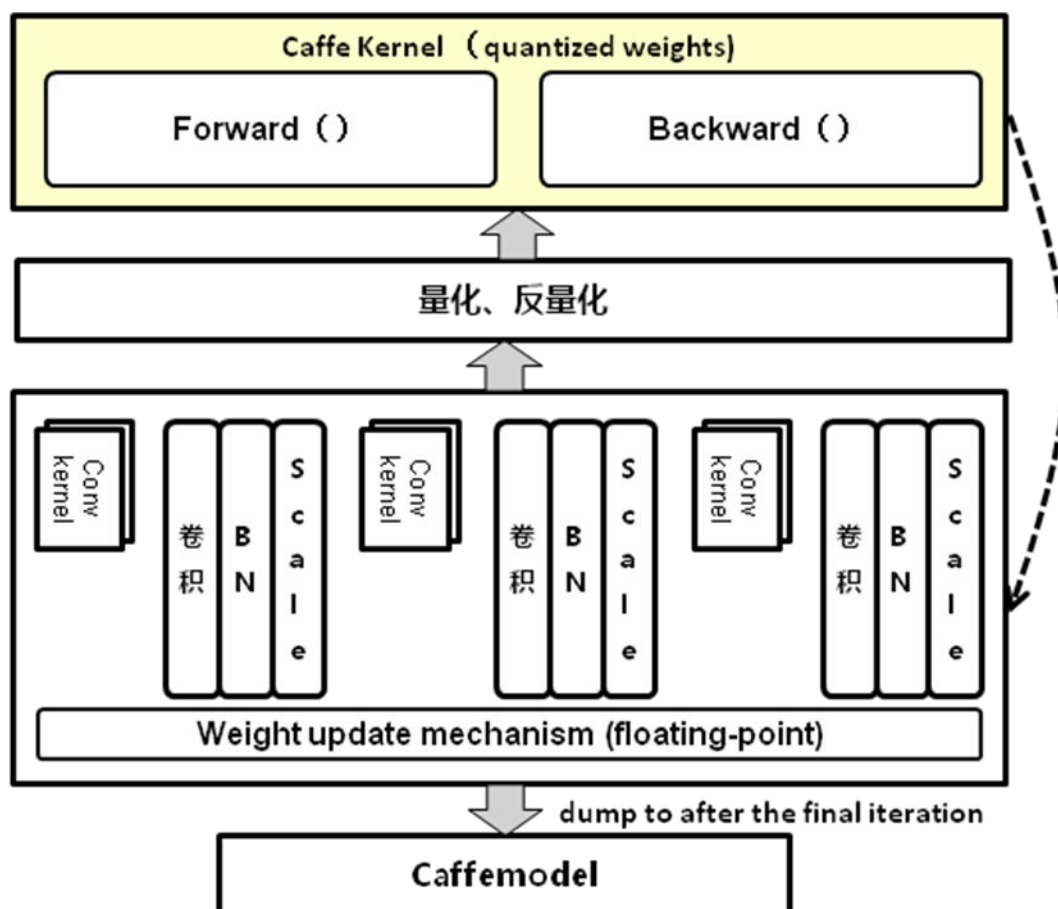
3.2.2 Finetune 训练时权重带量化的更新方法

权重的量化是将权重的连续取值近似为有限多个离散值的过程，离散值的二进制位数决定了量化的精度，NNIE定点运算的精度为8-bit。量化的过程是先将整个幅度划分成有限个小幅面（量化阶距）的集合，把落入某个阶距内的样值归为一类，并赋予相同的量化值。

从CNN网络finetune的角度（如3.2.1 权重量化的流程所述）来看，在每一次训练的迭代中，都会使用一个预先设置好的学习率，对各层的权重做调整，可以看出这个调整是渐进的，而当学习率或Caffe层梯度较小时，每次迭代对权重调整的幅度有可能比量化

阶距还要小，如果对每次迭代更新后的权重都立即做量化后复写CNN的网络模型，会导致模型的权重值“原地踏步”，无法收敛。

图 3-6 Finetune 训练过程

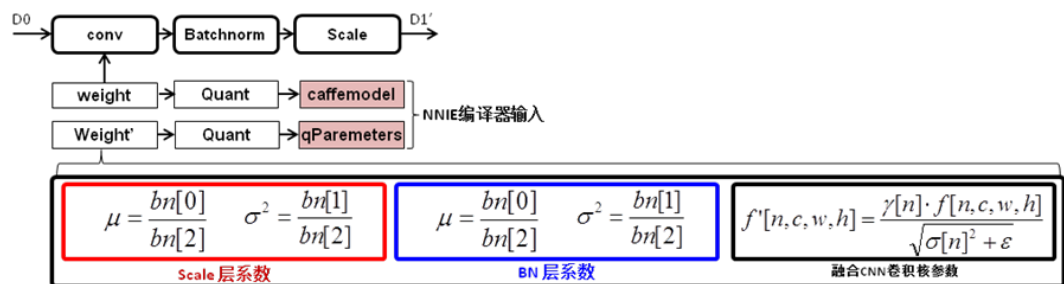


所以在网络finetune的时候需要备份一整套浮点的caffe模型，权重在训练过程中的每一次update都会在这套备份的浮点模型上完成，每次迭代将这套浮点模型量化后，做网络的forward和backward计算，详细操作见上图。

3.2.3 网络层融合（Layer-folding）时的权重量化方法

网络在训练时，很多情况下（特别是比较复杂的网络），卷积层后面会跟Batchnorm层和Scale层，对Feature Map做归一化，目的是为了网络更好地收敛。但是，在实际做Inference时，为了减少板端的计算量，通常会将卷积层、Batchnorm层和Scale层在数学上做融合。

图 3-7 Batchnorm 层和 Scale 层的融合



对于上述情况，做网络量化finetune是要注意浮点和定点混合操作的问题，需要注意这个问题的原因是，NNIE引擎的运算规格，对卷积、全连接等需要大量乘法运算的网络层做了定点运算的限制，Batchnorm和Scale层里乘法的运算量相对卷积较小，因此这些运算在NNIE引擎里是用高精度实现的。对于在训练的时候带着Batchnorm层和Scale层对网络finetune，在finetune训练的过程中，训练的过程中对卷积层做量化，对Batchnorm和Scale用浮点FP32运算，收敛后得到caffe模型（caffemodel），同时，在dump caffemodel的时候，对卷积、Batchnorm和Scale层融和后的权重weight做量化得到量化参数qParameters，将训练得到的caffemodel和qParameter给NNIE的编译器生成知识库。

3.3 训练

本小节简单介绍NNIE训练环境的搭建和训练的一些方法。

3.3.1 Finetune 训练环境的搭建

训练环境可以采用浮点的环境，具体搭建步骤如下：

- 步骤1** 按照前面章节的介绍将NNIE的量化库合入到caffe环境里。
- 步骤2** 重新编译caffe环境。
- 步骤3** 按照3.1的步骤和办法同时在训练和测试网络的prototxt中加上数据的量化层(QuantLayer)。
- 步骤4** 按照3.2.1和3.2.2的方法对训练脚本编程（可以参照海思release的量化Finetune的用例）。
- 步骤5** 把学习率适当调小，将浮点训练得到的收敛的网络模型做为初始模型。

----结束

3.3.2 Finetune 训练方法及注意事项

按照本文档仔细搭建环境和编写训练脚本，通常情况下由量化造成的精度损失都能通过微调（Finetune）恢复，量化模型能恢复到浮点模型的精度。如果重训后量化模型的精度无法恢复，或者重训后精度有所提升但是距离浮点的精度差距较大，可以尝试以下几种方法：

1. 只对数据做量化，不对weight做量化，如果精度可以恢复到浮点，我们基本上可以确定问题出在脚本。



2. 从数据层开始，自上而下逐层对浮点输出的Feature Map和量化Finetune之后输出的Feature Map做相似度比较，对于相似度较低（例如余弦相似度低于0.98的）的网络层逐层分析。
3. 训练脚本里在weight update的部分，注意把Batchnorm层和量化层排除掉，因为这些层的参数是统计出来的，不是学习得到的，不需要做update。

4 FAQ

4.1 量化库报错

量化库报错: Quantization by GPU(CUDA) failed(0x8: invalid device function)

图 4-1 量化库报错图

```
[ERR][check_cuda_result][192] Quantization by GPU(CUDA) failed(0x8: invalid device function)
[ERR][check_cuda_result][195] Please check GPU(CUDA) version. The library is built with '-gencode arch=compute_61,code=sm_61'
[ERR][_quant_and_dequant_gpu][319] ERROR: RUN FAILURE in /home/lenovo/100176142/gfpq/src/quant/quant_by_gpu.cu:319. Return = -65536
```

- 问题原因: GPU版本量化库不支持服务器的GPU架构, 只支持sm_30, sm_35, sm_37, sm_50, sm_52, sm_60, sm_61。
- 解决方法: 使用GFPQLibs_xxx.tgz里的库, 并执行get_gpu_version.sh, 确认服务器的CUDA gencode。

注意

Caffe训练网络示例里的GFPQLibs不一定是最新的, 请重新解压GFPQLibs.tgz到Caffe目录。

4.2 重训调用量化库报错

```
[ERR][HI_GFPQ_QuantAndDeQuant][105] Non linear quantization failed(0x2)
E0530 00:29:13.882642 34161 quant_layer.cu:30] HI_GFPQ_QuantAndDeQuant failed
```

- 问题原因: 量化库接口使用错误。
- 解决方法: 检查caffe/src/caffe/layers/quant_layer.cpp、caffe/src/caffe/layers/quant_layer.cu、wkQuant.py调用量化接口的代码, 是否参数不匹配。先执行sample代码, 确保能正常执行, 再参考sample代码修改。



注意

CPU接口传入CPU分配的内存，GPU接口传入GPU分配的内存。
